УДК 519.713

ТЕСТИРОВАНИЕ ПРОГРАММНОЙ РЕАЛИЗАЦИИ ПРОТОКОЛА IRC НА ОСНОВЕ МОДЕЛИ РАСШИРЕННОГО АВТОМАТА

М.В. Жигулин, А.В. Коломеец, Н.Г. Кушик, А.В. Шабалдин

Томский государственный университет
E-mail: maxzh81@gmail.com

Рассматривается тестирование программной реализации ngIRCd телекоммуникационного протокола IRC на основе теста, построенного по модели расширенного автомата. Компьютерные эксперименты иллюстрируют эффективность предложенного подхода.

Ключевые слова:
Протокол IRC, расширенный автомат, конечный автомат, проверяющий тест.

Key words:
IRC protocol, extended finite state machine, finite state machine, test suite.

Введение

Построение проверяющих тестов для телекоммуникационных протоколов является актуальной технической задачей. При тестировании технических систем, реализованных программно, на соответствие спецификации, проверяющие тесты часто строятся по модели расширенного автомата [1], который по семантике очень близок к программной реализации. Автомат содержит состояния, которые описывают различные «точки» программы, входные и выходные параметры и контекстные переменные. Переходы из состояния в состояние осуществляются при выполнении определенных условий, то есть при определенных значениях входных параметров и контекстных переменных. На каждом переходе контекстным переменным и выходным параметрам могут присваиваться новые значения. В данной работе мы показываем, что такой подход позволяет построить тесты, которые обнаруживают несоответствия в программных реализациях современных протоколов, а именно, в реализации протокола IRC (Internet Relay Chat).

1. Протокол IRC

Протокол IRC [2] позволяет организовать обмен текстовыми сообщениями между узлами сети Интернет в режиме реального времени. IRC функционирует в сети с архитектурой клиент-сервер, таким образом, обмен текстовыми сообщениями между клиентами производится через сервер.

Для подключения к IRC-серверу клиент должен идентифицировать себя. В том случае, когда разрешен вход без пароля, клиент передает две команды: *NICK* и *USER*, когда вход разрешен только с паролем, три команды: *PASS*, *NICK* и *USER*. В команде *PASS* сообщается пароль пользователя, в команде *NICK* – псевдоним, в команде *USER* – имя и параметры пользователя (имя пользователя может совпадать с псевдонимом, но не обязательно). После успешного прохождения идентификации пользователь считается зарегистрированным и может подключиться к одному из доступных каналов на сервере с помощью команды *JOIN* или создать новый канал. Общение пользователей друг с другом происходит в рамках канала: один пользователь может передать сообщение другому пользователю, подключенному к тому же каналу. Для окончания сеанса связи клиент отправляет команду *QUIT*.

2. Тестирование протокола IRC на соответствие спецификации

Для построения тестов по спецификации протокола было построено два расширенных автомата. Первый расширенный автомат соответствует случаю, когда для клиента разрешен вход без пароля (рис. *а*); второй расширенный автомат предусматривает строгую необходимость указания пароля при идентификации клиента (рис. *б*). Расширенный автомат на рис. *а*, имеет 4 состояния, 52 перехода, 12 входных и 25 выходных символов, 6 контекстных переменных и 17 входных и выходных параметров. Каждый из переходов может быть выполнен при определенных значениях входных параметров и контекстных переменных. Для построения проверяющего теста поведение каждого из двух расширенных автоматов описывается соответствующим конечным автоматом; по полученным конечным автоматам строятся проверяющие тесты на основе обхода графа переходов [3]. Такие тесты гарантированно обнаруживают все выходные неисправности, и как будет показано далее, обнаруживают и ряд других несоответствий в протокольной реализации.

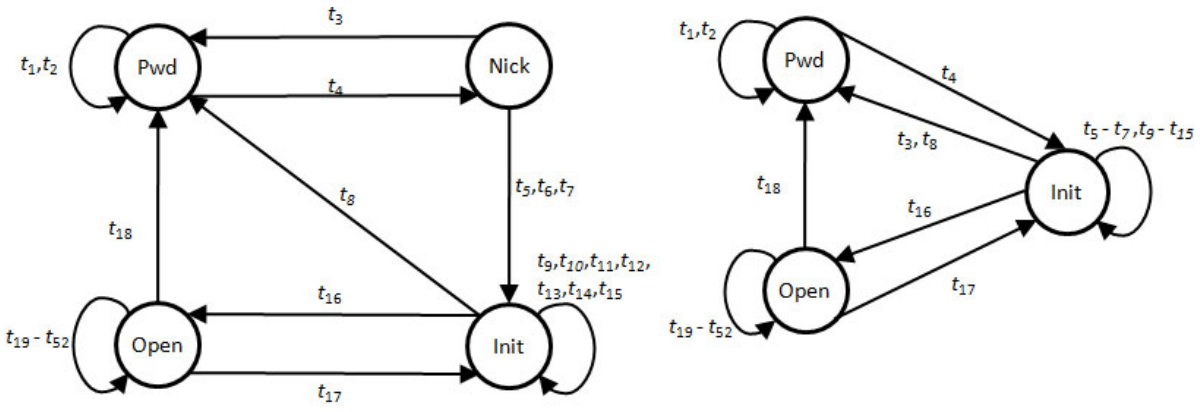
В качестве примера рассмотрим некоторые переходы из состояния *Pwd* (переходы с номерами 1, 2, 4 на рис. *а*).

t_1: *Pwd*, *PASS* (*Password*),(*Password*==1 and *ConnectPassword*==1) / 461, *Pwd*

t_2: *Pwd*, *PASS* (*Password*),(*ConnectPassword*!=1) / 462, *Pwd*

t_4: *Pwd*, *PASS* (*Password*), (*Password*!=1 and *ConnectPassword*==1) / *NULL*, *ConnectPassword*= *Password*, *Nick*

В состоянии *Pwd* на вход автомата подается входной символ *PASS* с параметром *Password* и в зависимости от значения параметра *Password* и значе-



а *б*

Рисунок. *Расширенный автомат IRC протокола. Разрешен вход: а) без пароля; б) только с паролем*

ния контекстной переменной *ConnectPassword*, которая содержит информацию о состоянии авторизации, автомат остается либо в том же состоянии, либо переходит в состояние *Nick*. Если предикат (*Password*==1 and *ConnectPassword*==1) истинен, автомат остается в состоянии *Pwd* с выходом 461.

3. Построение эквивалентного конечного автомата

Чтобы определить поведение расширенного автомата под действием параметризированной входной последовательности, т е. последовательности, состоящей из входных символов с конкретными значениями параметров, поведение расширенного автомат моделируется на этой последовательности. Для заданного состояния s автомата M, вектора $\mathbf{v}$ значений контекстных переменных, входного символа x и вектора $\boldsymbol{\rho}$ значений входных параметров определяется переход (s, x, P, op, y, up, s') из состояния s, на котором предикат P является истинным для входного вектора $\boldsymbol{\rho}$ и контекстного вектора $\mathbf{v}$. На следующем шаге согласно функциям данного перехода вычисляется новый контекстный вектор $\mathbf{v}'$, определяются следующая конфигурация $s'\mathbf{v}'$ и параметризированный выходной символ $y\boldsymbol{\omega}$. Если области определения входных параметров и контекстных переменных конечны, то после полного моделирования можно построить конечный автомат, эквивалентный расширенному автомату M.

Состояниями конечного автомата являются конфигурации $s\mathbf{v}$, достижимые из начальной конфигурации. Входными и выходными символами конечного автомата являются соответственно параметризированные входные и выходные символы расширенного автомата. Соответственно число состояний конечного автомата, если такой автомат существует, равно числу различных конфигураций, достижимых из начальной конфигурации расширенного автомата. В данной работе мы рассматриваем непротиворечивые и полностью определенные расширенные автоматы. В частности, мы полагаем, что в любой конфигурации для любого параметризированного входного символа существует единственный переход, предикат которого принимает значение «ИСТИНА». Соответственно конечный автомат, эквивалентный такому расширенному автомату (если такой конечный автомат существует), является детерминированным.

Более того, мы полагаем, что для каждой достижимой конфигурации $s\mathbf{v}$ в расширенном автомате и каждого параметризированного входного символа $x\boldsymbol{\rho}$ существует переход из состояния s по входному символу x, предикат которого равен «ИСТИНА» при значениях контекстных переменных из вектора $\mathbf{v}$ и значениях входных параметров из вектора $\boldsymbol{\rho}$, т. е. соответствующий конечный автомат (если существует) является полностью определенным.

Заметим, что даже в случае, когда эквивалентный конечный автомат можно построить, т. е. в случае, когда области определения контекстных переменных и входных параметров конечны, построенный конечный автомат имеет намного больше состояний, чем исходный расширенный автомат, так как частью состояния эквивалентного конечного автомата является вектор значений контекстных переменных, а частью входного символа – вектор значений входных параметров. Таким образом, размерность соответствующего конечного автомата оказывается очень большой, и построить проверяющий тест по такому автомату достаточно сложно. Поэтому обычно при моделировании поведения расширенного автомата вводится ограничение на число состояний и/или переходов соответствующего эквивалентного автомата. В этом случае построенный конечный автомат только частично описывает поведение моделируемого конечного автомата. Более того, в случае, когда области значений некоторых контекстных переменных и/или входных параметров бесконечны, то эквивалентный конечный автомат полностью построить нельзя.

Ниже приведена расшифровка некоторых входных и выходных символов, которые используются в конечных автоматах, полученных при моделировании расширенных автоматов, описывающих поведение протокола IRC.

- *PASS*(1) – команда *PASS* с пустым паролем;
- *PASS*(2) – команда *PASS* с правильным непустым паролем;
- *NULL* – пустой символ, означает отсутствие ответа;
- *NICK*(1) – команда *NICK* с пустым псевдонимом;
- *NICK*(3) – команда *NICK* с непустым псевдонимом;
- *USER*(1,0,5) – команда *USER* с пустым именем пользователя;
- *USER*(3,0,5) – команда *USER* с непустым именем пользователя;
- *MODE*(1,7) – команда *MODE* (задает режим работы пользователя) с пустым псевдонимом и режимом +*i* (невидимость);
- *MODE*(3,7) – команда *MODE* с непустым псевдонимом и режимом +*i*;
- *QUIT*(6) – команда завершения сеанса работы с сервером с некоторым произвольным сообщением;
- *JOIN*(1,1) – команда *JOIN* (подключение к каналу) с пустыми параметрами;
- *JOIN*(35,1) – команда *JOIN* подключения к некоторому существующему каналу на сервере;
- *JOIN*(36,1) – команда *JOIN* подключения к несуществующему каналу на сервере;
- *JOIN*(37,1) – команда *JOIN* подключения к каналу с недопустимым именем;
- *PART*(1), *PART*(35), *PART*(36), *PART*(3) – команды *PART* выхода из всех каналов, из существующего канала, из несуществующего канала, из другого существующего канала соответственно;
- *NAMES*(1), *NAMES*(35), *NAMES*(36), *NAMES*(3) – команды *NAMES* вывода псевдонимов подключенных пользователей на неуказанном канале, на существующем канале, на несуществующем канале, на другом существующем канале соответственно;
- *LIST*(1), *LIST*(35), *LIST*(36), *LIST*(3) – команды *LIST* вывода имен всех каналов, имени некоторого существующего канала, имени несуществующего канала, имени другого существующего канала соответственно;
- *TOPIC*(1), *TOPIC*(35), *TOPIC*(36), *TOPIC*(3) – команды *TOPIC* вывода темы неуказанного канала, некоторого существующего канала, несуществующего канала, другого существующего канала соответственно;
- *PRIVMSG*(1,1) – команда *PRIVMSG* отправки пустого персонального сообщения неуказанному пользователю;
- *PRIVMSG*(3,1) – команда *PRIVMSG* оправки пустого персонального сообщения существующему пользователю;
- *WHO*(41) – команда *WHO* получения информации обо всех пользователях, подключенных к текущему серверу;
- *WHO*(42) – команда *WHO W** получения информации обо всех пользователях, подключенных к текущему серверу, чей псевдоним начинается на букву *W*;
- 403, 411, 412, 442, 502, 461 и т. д. – сообщения сервера с одним из указанных кодов ответа;
- *ANY* – произвольный непустой ответ сервера, заканчивающийся символами перехода на новую строку.

В таблице представлен фрагмент таблицы переходов/выходов конечного автомата, описывающего поведение расширенного автомата на рисунке *a*. Строки таблицы соответствуют параметризированным входным символам; столбцы соответствуют конфигурациям, достижимым из начальной конфигурации. Построенный конечный автомат имеет 9 состояний и 34 входных символа. Пустые клетки в таблице соответствуют переходу автомата в то же состояние с выдачей специального сигнала ошибки или сигнала *NULL*.

4. Построение теста

По полученным конечным автоматам были построены тесты на основе обхода графа переходов, где под *тестом* понимается множество входо-выходных последовательностей конечного автомата. Для каждой пары (*s*, *i*) («состояние, входной символ») тест, построенный на основе обхода графа переходов, содержит входную последовательность αi, где последовательность α переводит конечный автомат из начального состояния в состояние *s*. Иными словами, такой тест фактически «проходит» по каждому ребру графа переходов автомата, и как отмечалось выше, обнаруживает все выходные ошибки. Полученные тесты были далее объе-

Таблица. *Фрагмент конечного автомата, эквивалентного расширенному автомату*

	Pwd(1,1,1,0,1,1)	*Nick*(2,1,1,0,1,1)	*Init*(2,1,1,0,1,1)	*Init*(2,3,1,0,1,1)
PASS(1)	*Pwd*(1,1,1,0,1,1)/461		*Init*(2,1,1,0,1,1)/462	*Init*(2,3,1,0,1,1)/462
PASS(2)	*Nick*(2,1,1,0,1,1)/*NULL*		*Init*(2,1,1,0,1,1)/462	*Init*(2,3,1,0,1,1)/462
NICK(1)		*Init*(2,1,1,0,1,1)/431	*Init*(2,1,1,0,1,1)/431	*Init*(2,3,1,0,1,1)/ *STR_NICK*
NICK(3)		*Init*(2,3,1,0,1,1)/ *NULL*	*Init*(2,3,1,0,1,1)/NULL	*Init*(2,3,1,0,1,1)/433
NICK(43)			*Init*(2,1,1,0,1,1)/ *ERR_ERRONEUSNICKNAME*	*Init*(2,3,1,0,1,1)/*ERR_ERRONEUSNICKNAME*
USER(1,0,5)			*Init*(2,1,1,0,1,1)/461	*Init*(2,3,1,0,1,1)/461
USER(3,0,5)			*Open*(2,1,3,0,5,1)/1	*Open*(2,3,3,0,5,1)/1

динены в единый тест, который и использовался для тестирования реализации протокола *IRC*. Тест, построенный обходом графа переходов первого конечного автомата, содержит 265 тестовых последовательностей. Общая длина теста с учетом того, что перед каждой тестовой последовательностью подается входной символ *QUIT* (6), возвращающий реализацию в начальное состояние, составляет 1164 входных символов. Тест для второго автомата содержит 298 тестовых последовательностей; общая длина теста с учетом сигнала СБРОС составляет 1495. Общая длина теста после объединения тестов для двух автоматов и удаления последовательностей, которые являются собственными префиксами других последовательностей, равна 1164.

5. Экспериментальные результаты

Построенный тест был помещен в базу тестов программы *Tester* [4]. В качестве тестируемой реализации была выбрана свободно распространяемая программа с открытыми исходными текстами *ngIRCd* последней версии (16) [5], являющаяся распространенной реализацией сервера IRC. Программа была загружена с сайта разработчиков в виде исходных текстов и скомпилирована с использованием опции *strict-rfc*, которая создает версию, соответствующую спецификации RFC.

Для тестирования программа *ngIRCd* была запущена со стандартными настройками на том же компьютере, на котором работал тестер-клиент программы *Tester*. В параметрах тестера был выбран порт 6667 – стандартный порт IRC-сервера.

В процессе тестирования входные воздействия из тестовых последовательностей подавались один за другим в том же порядке, как и в построенном тесте. После окончания каждой тестовой последовательности подавалась сбрасывающая последовательность, состоящая из единственной команды *QUIT*. После подачи каждого входного воздействия в течение 3 с ожидался ответ от сервера. Согласно эксперементальным данным, указанный период ожидания ответа является максимальным временем, за которое тестируемая реализация предоставляет полный ответ на запрос. Таким образом, если в указанный промежуток времени ответ не пришел, то полагалось, что ответ – пустой. Если полученный ответ не соответствовал ожидаемому согласно спецификации ответу, подача тестовой последовательности прекращалась и фиксировалась ошибка.

В результате экспериментов в тестируемой реализации было обнаружено 3 ошибки. На следующих тестовых последовательностях были обнаружены ошибки: *PASS* (2) / *NULL NICK* (1) / 431; *PASS* (2) / *NULL NICK* (3) / *NULL USER* (3,0,5) / 001 NICK (3) / 433; *PASS* (2) / *NULL NICK* (3) / *NULL USER* (3,0,5) / 001 *MODE* (1,7) / 461.

Первая ошибка заключается в неправильном коде ответа на команду *NICK* с пустым параметром (т. е. без указанного псевдонима). Согласно спецификации [RFC2812, section 3.1.2 (раздел 3.1.2)], ответом на *NICK* без параметра должно быть сообщение с кодом 431 (*ERR_NONICKNAMEGIVEN*), сервер же возвращает код ответа 461 (*ERR_NEEDMOREPARAMS*).

Вторая ошибка заключается в неправильной обработке сервером попытки использования уже занятого псевдонима. В том случае, когда псевдоним, который передается в параметре команды *NICK*, уже занят, сервер должен вернуть ответ с кодом 433 (*ERR_NICKNAMEINUSE*). Тестируемая реализация в ответ на попытку повторной регистрации уже используемого псевдонима вернула пустое сообщение.

Последняя обнаруженная ошибка является следствием неправильной обработки команды *MODE* с отсутствующим псевдонимом, но в которой присутствует второй параметр (+*i*), задающий режим работы. Согласно спецификации [RFC2812, section 3.1.5 (раздел 3.1.5)] ответом на такую команду может быть сообщение с кодом 502 (*ERR_USERSDONTMATCH*) или с кодом 461 (*ERR_NEEDMOREPARAMS*). Тестируемый сервер вернул сообщение с кодом 401 (*ERR_NOSUCHNICK*).

Заключение

Показано, что тесты по программной реализации ngIRCd телекоммуникационного протокола IRC, построенные по модели расширенного автомата, позволяют обнаруживать «несоответствия» в программных реализациях телекоммуникационных протоколов относительно спецификации.

Описание в виде расширенного автомата достаточно близко к описанию различных программных реализаций, поэтому данные методы могут быть использованы при синтезе тестов для проверки программного обеспечения.

Авторы выражают особую благодарность руководителю проекта, проф. Н.В. Евтушенко за полезные научные дискуссии, а также аспиранту РФФ ТГУ А.Л. Никитину за разрешение использовать его программные реализации при проведении компьютерных экспериментов. Работа частично поддержана проектом ФЦП (государственный контракт № 02.514.12.402).

СПИСОК ЛИТЕРАТУРЫ

1. El-Fakih K., Prokopenko S., Yevtushenko N., Bochmann G. Fault diagnosis in extended finite state machines // In Proc. of the IFIP 15th Intern. Conf. on Testing of Communicating Systems (TestCom2003): LNCS. – France, 2003. – V. 2644. – P. 197–210.
2. RFC2812 – Internet Relay Chat: Client Protocol. 2000. URL: http://www.faqs.org/rfcs/rfc2812.html (дата обращения: 10.09.2010).
3. Евтушенко Н.В., Петренко А.Ф., Ветрова М.В. Недетерминированные автоматы: анализ и синтез. Ч. 1. Отношения и операции. – Томск: Том. гос. ун-т, 2006. – 142 с.
4. Shabaldin A. Constructing a Tester for Checking Student Protocol Implementations // In: Proc. of SYRCoSE 2007. – Moscow, Russia, 2007. – V. 2. – P. 23–29.
5. ngIRCd: Next Generation IRC Daemon. 2010. URL: http://ngircd.barton.de (дата обращения: 10.09.2010).

Поступила 01.11.2010 г.